

# *Asynchronous Checkpoint Migration with MRNet in the Scalable Checkpoint/Restart Library*

FTXS Workshop  
June 25, 2012

Kathryn Mohror



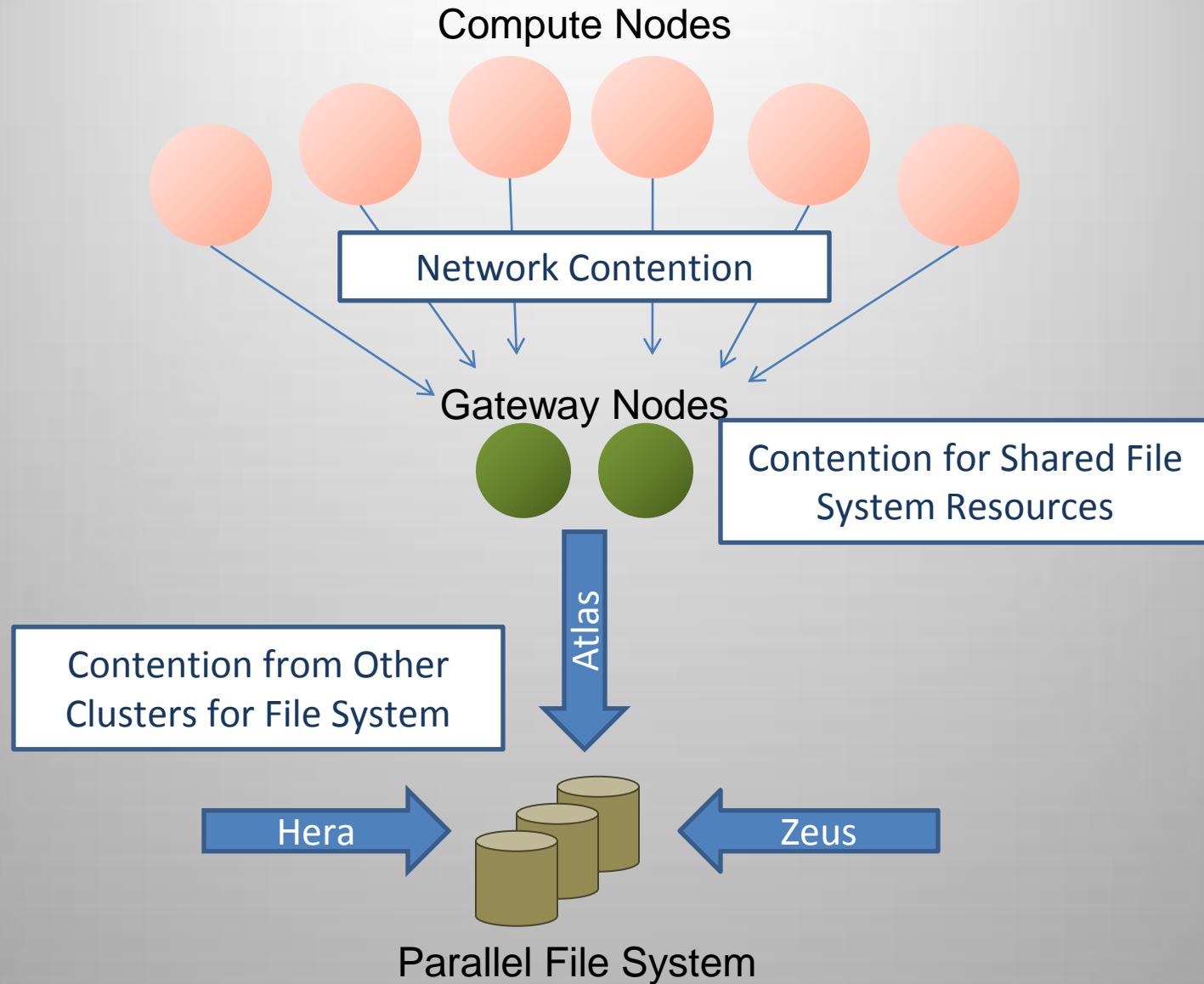
Lawrence Livermore  
National Laboratory

LLNL-PRES-562317

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC



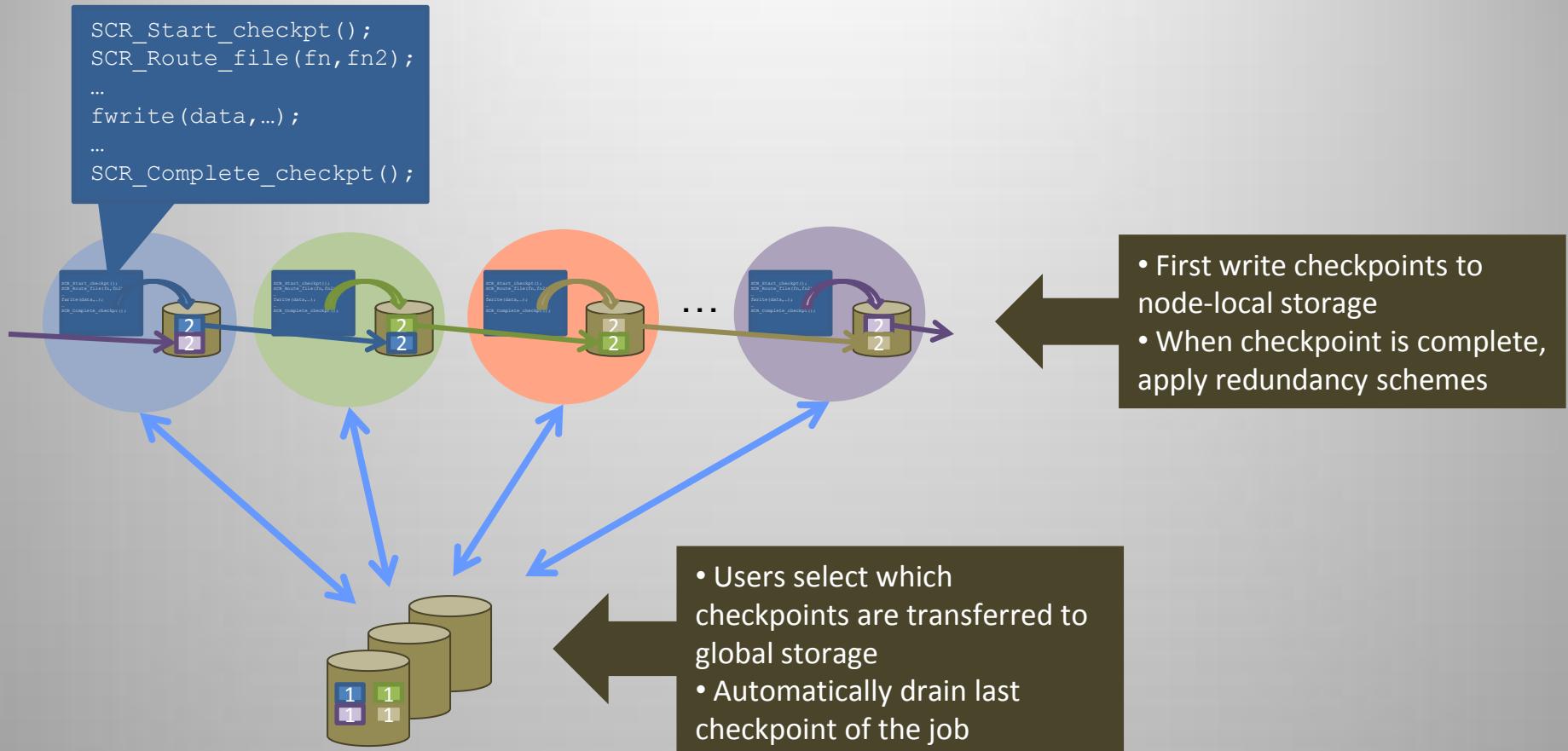
# Writing checkpoints to the parallel file system is very expensive



# SCR uses node-local storage to reduce checkpointing costs

- Observations:
  - Only need the most recent checkpoint data.
  - Typically just a single node failed at a time.
- Idea:
  - Store checkpoint data redundantly on compute cluster; only write a few checkpoints to parallel file system.
- Node-local storage is a performance opportunity AND challenge
  - + Scales with rest of system
  - - Fails and degrades over time
  - - Physically distributed
  - - Limited resource

# SCR utilizes node-local storage and the parallel file system



# Make a few simple function calls to use SCR

```
int main(int argc, char* argv[]) {
    MPI_Init(argc, argv);

    for(int t = 0; t < Timesteps; t++)
    {
        /* ... Do work ... */

        checkpoint();
    }

    MPI_Finalize();
    return 0;
}
```

```
void checkpoint() {

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    char file[256];
    sprintf(file, "rank_%d.ckpt", rank);

    FILE* fs = fopen(file, "w");
    if (fs != NULL) {
        fwrite(state, ..., fs);
        fclose(fs);
    }

    return;
}
```

# Make a few simple function calls to use SCR

```
int main(int argc, char* argv[]) {
    MPI_Init(argc, argv);
    SCR_Init();

    for(int t = 0; t < Timesteps; t++)
    {
        /* ... Do work ... */

        int flag;
        SCR_Need_checkpoint(&flag);
        if (flag)
            checkpoint();
    }

    SCR_Finalize();
    MPI_Finalize();
    return 0;
}
```

```
void checkpoint() {
    SCR_Start_checkpoint();

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    char file[256];
    sprintf(file, "rank_%d.ckpt", rank);

    char scr_file[SCR_MAX_FILENAME];
    SCR_Route_file(file, scr_file);
    FILE* fs = fopen(scr_file, "w");
    if (fs != NULL) {
        fwrite(state, ..., fs);
        fclose(fs);
    }

    SCR_Complete_checkpoint(1);
    return;
}
```

# Make a few simple function calls to use SCR

```
int main(int argc, char* argv[]) {
    MPI_Init(argc, argv);
    SCR_Init();

    for(int t = 0; t < Timesteps; t++)
    {
        /* ... Do work ... */

        int flag;
        SCR_Need_checkpoint(&flag);
        if (flag)
            checkpoint();
    }

    SCR_Finalize();
    MPI_Finalize();
    return 0;
}
```

```
void checkpoint() {
    SCR_Start_checkpoint();

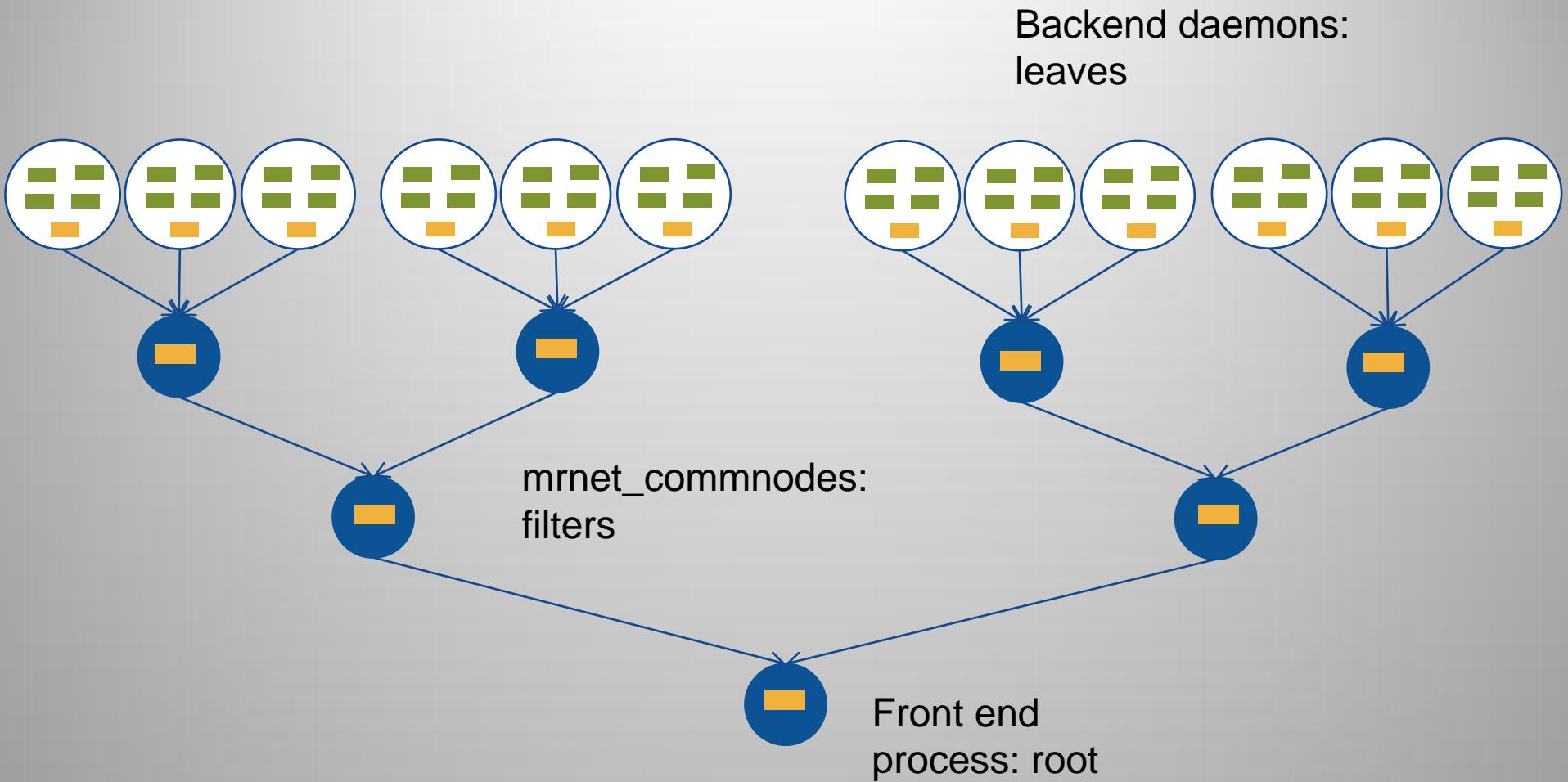
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    char file[256];
    sprintf(file, "rank_%d.ckpt", rank);

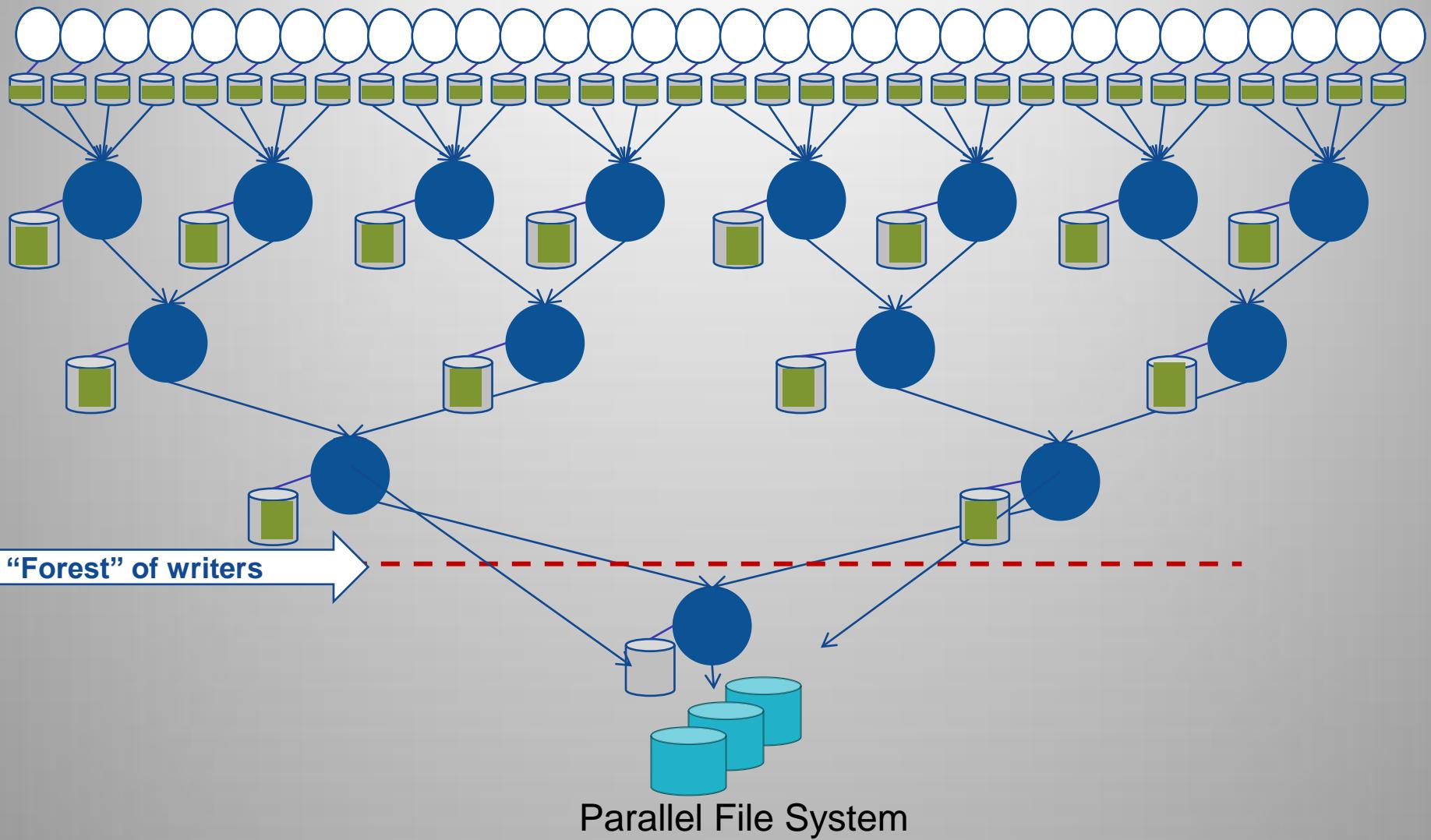
    char scr_file[SCR_MAX_FILENAME];
    SCR_Route_file(file, scr_file);
    FILE* fs = fopen(scr_file, "w");
    if (fs != NULL) {
        fwrite(state, ..., fs);
        fclose(fs);
    }

    SCR_Complete_checkpoint(1);
    return;
}
```

# MRNet Instance



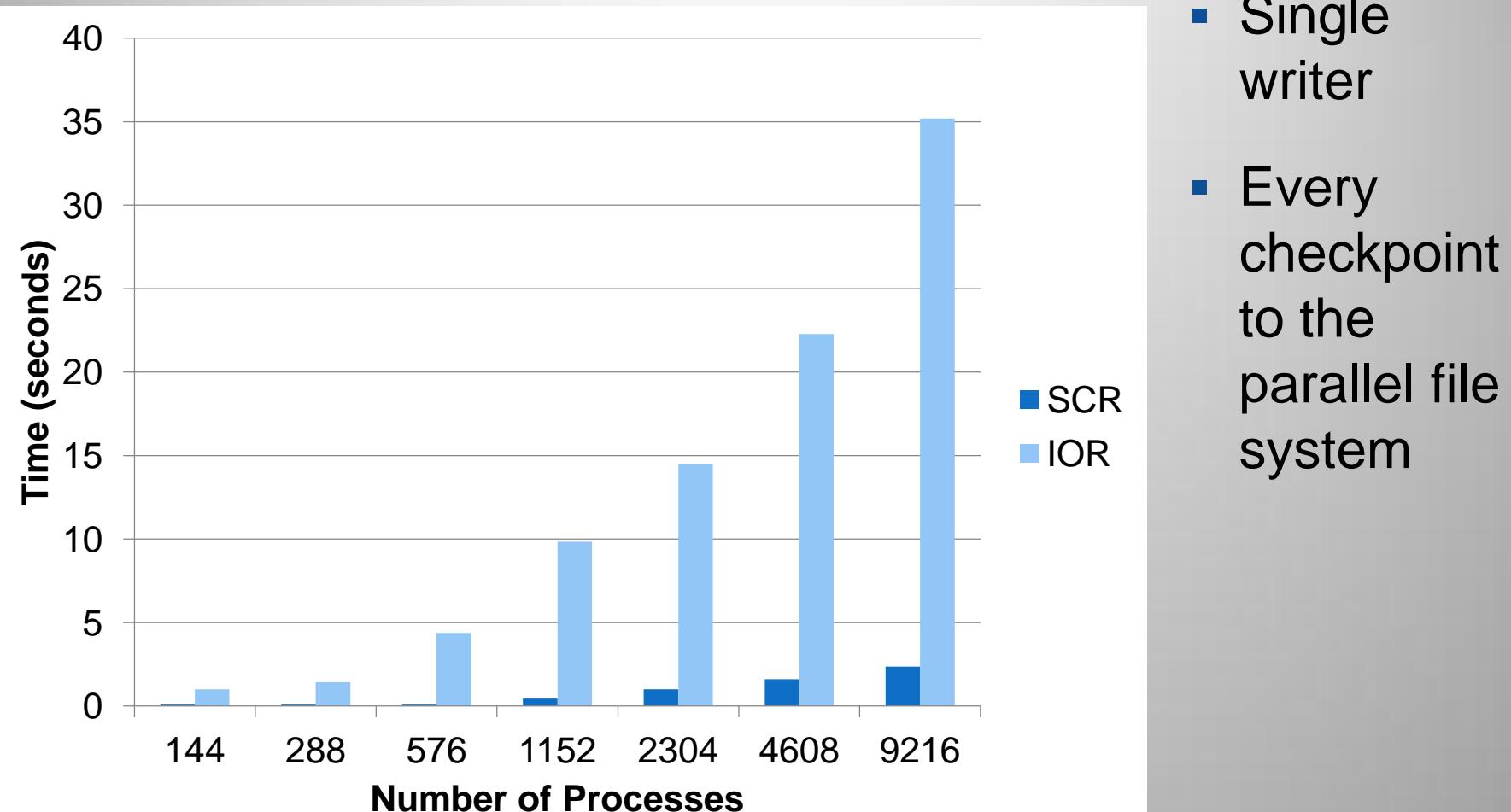
# SCR/MRNet integration writes checkpoints to the PFS in a controlled way



# Experimental setup

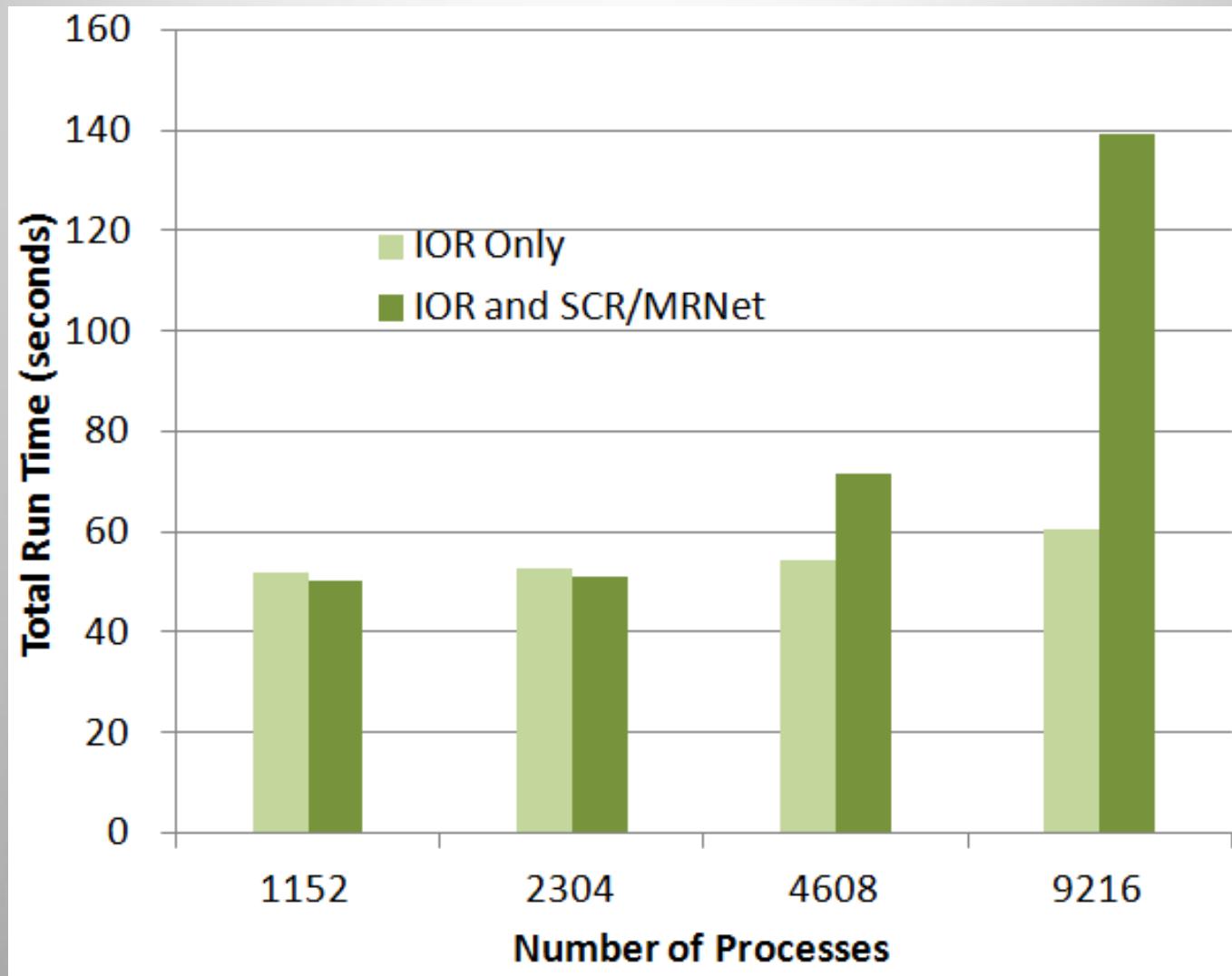
- Sierra
  - 1944 node Linux cluster, 12 cores per node
- Lustre
  - Lscratchc, 1.6 TB, 30 GB/s
- MRNet version 3.0
- LaunchMON version 0.7.2
- SCR 1.1-8
- IOR benchmark
- MRNet fanout of 48
  - 48 backend daemons for every mrnet\_commnnode
- MRNet processes run on additional compute nodes

# Average Total I/O Time per checkpoint with and without SCR/MRNet



- Single writer
- Every checkpoint to the parallel file system

# Total Run Time



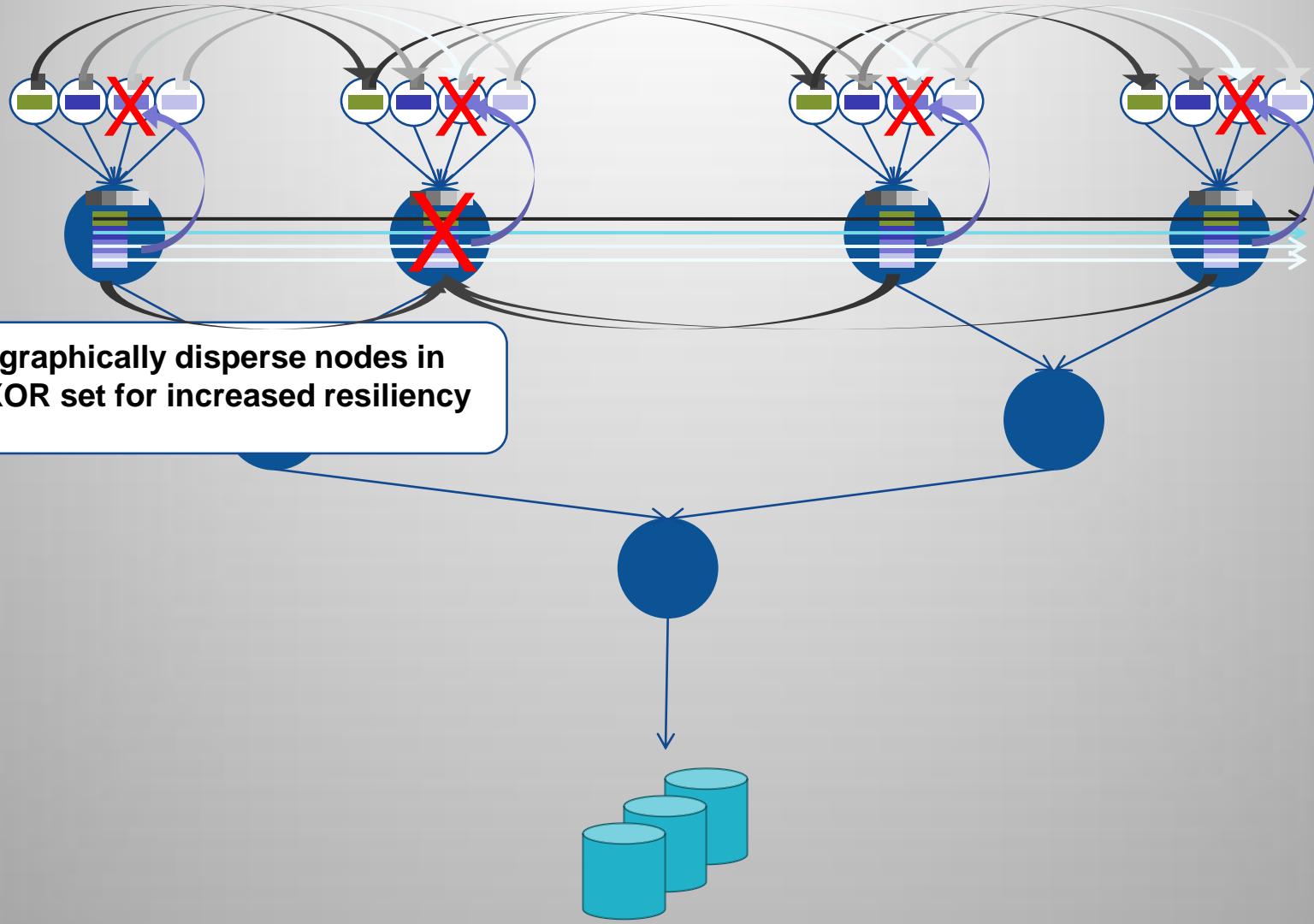
# Scalability Issues

- SCR\_Finalize writes final checkpoint to PFS, is a synchronous operation
- Front end process as only writer is a bottleneck, causes application to block in SCR\_Complete\_checkpoint
  - Optimization: forest of writers
- Perturbation of application by back end daemons
  - Intercept I/O operations
  - RDMA transfer of checkpoints

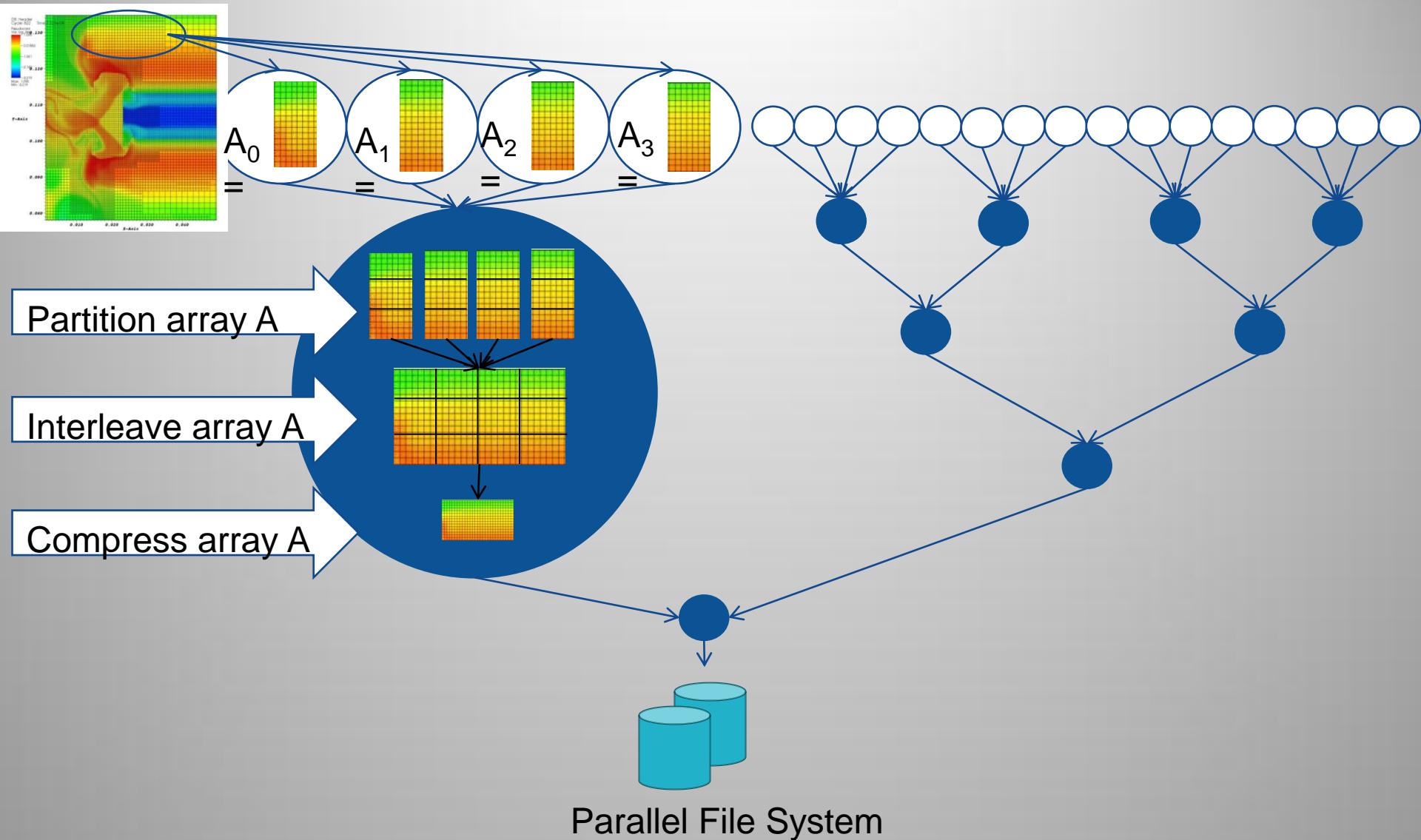
# Conclusions

- Asynchronous checkpoint movement with SCR/MRNet is successful in reducing the I/O time of applications
- However, scalability problems remain
  - Forest of writers
  - Optimize backend daemons to minimize perturbation
  - Or better, intercept writes and use RDMA transfer
  - Optimize MRNet tree depth and fanout

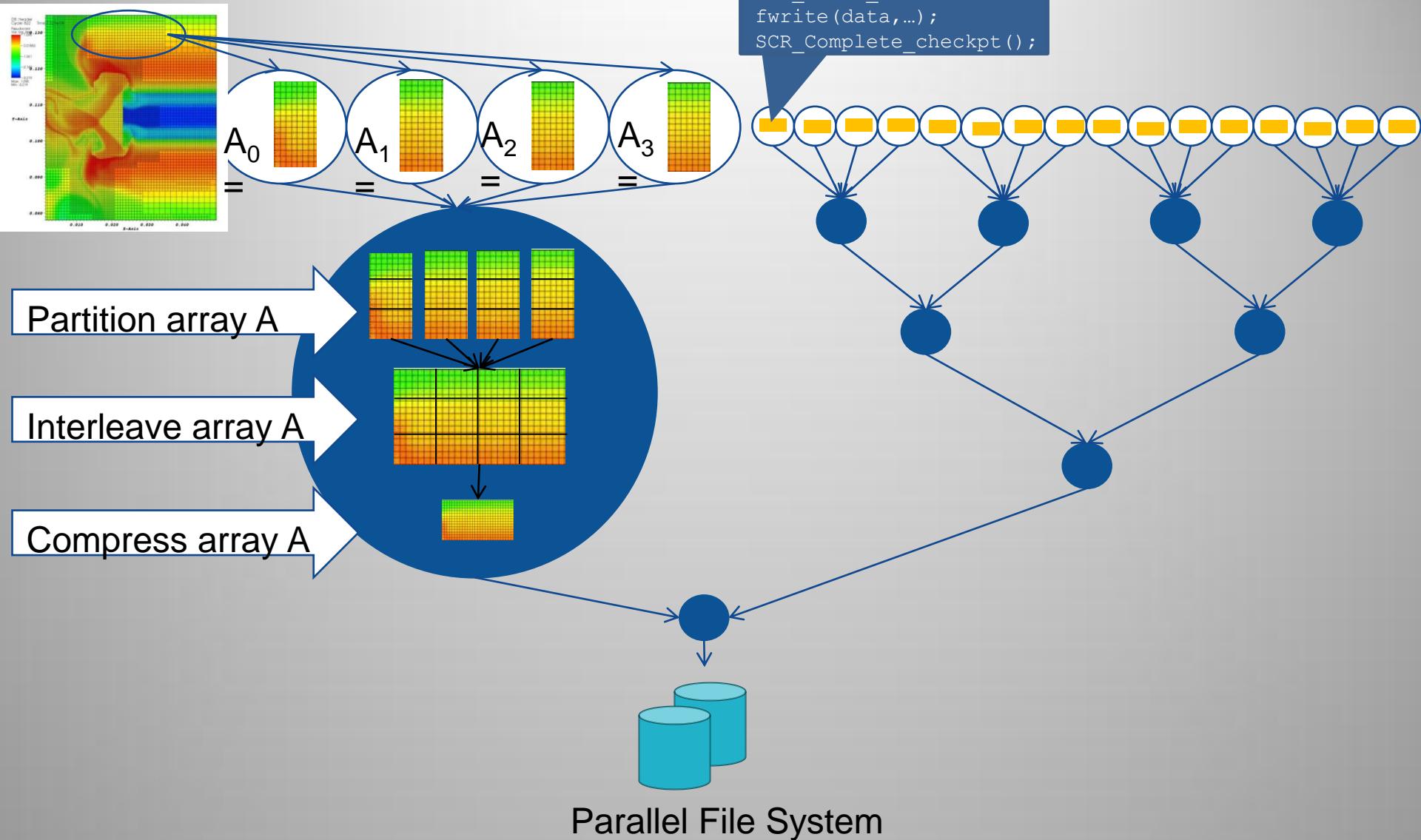
# Future work: additional levels of resiliency



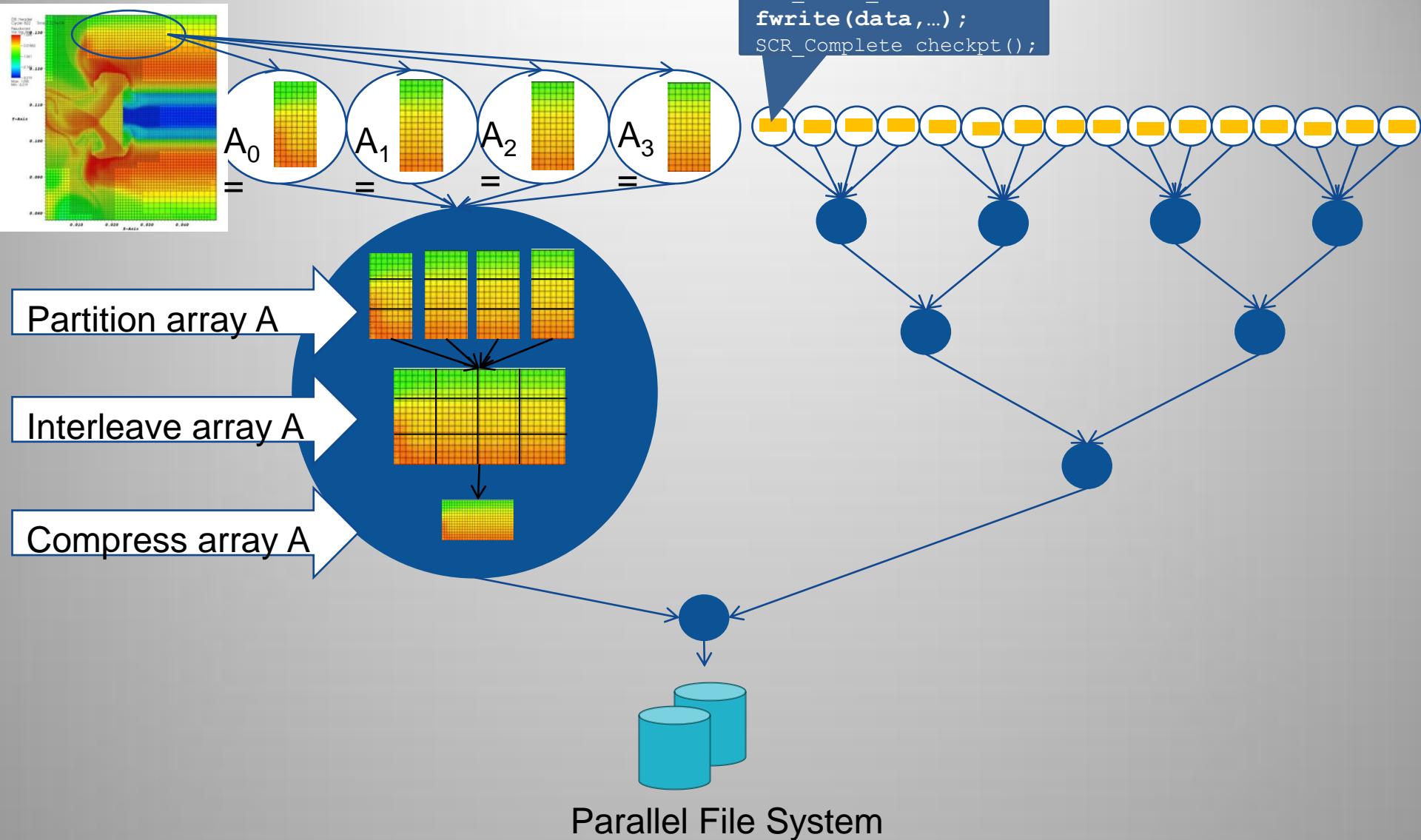
# Future work: compress checkpoints to reduce checkpointing overheads



# Future work: Intercept I/O operations for additional optimizations



# Future work: Intercept I/O operations for additional optimizations



# Thanks!

- Adam Moody, Bronis de Supinski (LLNL)
- *Checkpoint compression*: Tanzima Islam, Saurabh Bagchi, Rudolf Eigenmann (Purdue)
- *RDMA checkpoint transfer*: Kento Sato, Naoya Maruyama, Satoshi Matsuoka (Tokyo Tech)
- *Intercept file I/O, ensemble checkpointing*: Raghunath Raja Chandrasekar, Dhabaleswar (DK) Panda (The Ohio State University)
- For more information
  - kathryn@llnl.gov, moody20@llnl.gov
  - Open source, BSD license:  
<http://sourceforge.net/projects/scalablecr>



**Lawrence Livermore  
National Laboratory**